Exhibit C

```
// Useful methods for converting and comparing colors
//////////////////////////////////////////////////////////////////////

#ifndef _COLOR_LOGIC_INCLUDE
#define _COLOR_LOGIC_INCLUDE

#include <math.h>

///////////////////////////////////////////////////////////////////////////
// **  NOTE  **  NOTE  **  NOTE  **  NOTE  **  NOTE  **  NOTE  **  NOTE  **
///////////////////////////////////////////////////////////////////////////
// [redacted]
//
//
//
//
//
//
//
//
//
// This new method uses HSB space to determine if two colors are "too close"
// to each other be visible to on a CRT or LCD monitor.  The value ranges
// that are used are based on several weeks of testing hundreds of monitors.
// [redacted]
///////////////////////////////////////////////////////////////////////////
// **  NOTE  **  NOTE  **  NOTE  **  NOTE  **  NOTE  **  NOTE  **  NOTE  **
///////////////////////////////////////////////////////////////////////////

namespace ColorLogic
{
        // Hue/Saturation/Brightness struct
        typedef struct tagHSB
        {
                int nHue;                       // Degree (0-360)
                int nSaturation;        // Percentage (0-100)
                int nBrightness;        // Percentage (0-100)
        } HSB;

        // Round a double to the given precision
        // Used in RGB->HSB conversion method
        double DblRound(double dValue, int dPrecision)
        {
                static const double dBase = 10.0f;
                double dComplete5, dComplete5i;

                dComplete5 = dValue * pow(dBase, (double)(dPrecision + 1));

                if(dValue < 0.0f)
                        dComplete5 -= 5.0f;
                else
                        dComplete5 += 5.0f;

                dComplete5 /= dBase;
                modf(dComplete5, &dComplete5i);

                return dComplete5i / pow(dBase, (double)dPrecision);
        }
        // returns the difference between the min and the max
        int minmax(int i1, int i2)
        {
                return max(i1, i2) - min(i1,i2);
        }

        // Converts the given RGB color to Hue/Saturation/Luminance
        // Note: Photoshop seems to floor values instead of rounding...
        //       rounding is more accurate
        void RGB_to_HSB(COLORREF crRGB, HSB& hsb)
        {
                WORD wRed = GetRValue(crRGB);
                WORD wGreen = GetGValue(crRGB);
                WORD wBlue = GetBValue(crRGB);

                // Find the min and max RGB values
                WORD wMax = max(wRed, max(wGreen, wBlue));
                WORD wMin = min(wRed, min(wGreen, wBlue));

                // Calculate the brightness
                hsb.nBrightness = (int)DblRound((((double)wMax * 100) / 255), 0);

                // If this is grey we are done
                if(wMax == wMin)
                {
                        hsb.nHue = 0;
                        hsb.nSaturation = 0;
                }
                else
                {
                        // Calculate the saturation
                        hsb.nSaturation = (int)DblRound((((double)100 * (wMax -
wMin)) / wMax), 0);

                        // Calculate the hue
                        double dDiff = wMax - wMin;
                        double dR = (wMax - wRed) / dDiff;
                        double dG = (wMax - wGreen) / dDiff;
                        double dB = (wMax - wBlue) / dDiff;
                        double dHue = 0;
                        if(wRed == wMax)
                                dHue = dB - dG;
                        else if(wGreen == wMax)
                                dHue = 2 + dR - dB;
                        else if(wBlue == wMax)
                                dHue = 4 + dG - dR;

                        hsb.nHue = (int)DblRound((dHue * 60) + 360, 0) % 360;
                }
        }

        bool IsColorVisible(COLORREF crFG, COLORREF crBG)
        {
                // RGB->HSB conversions
                HSB hsbFG, hsbBG;
                RGB_to_HSB(crFG, hsbFG);
                RGB_to_HSB(crBG, hsbBG);

                bool bVisible = true;
                // the hue is in degrees and can wrap around, so we find
                // the shortest distance between the two colors
                int nHueDiff = abs(hsbFG.nHue - hsbBG.nHue);
                if(nHueDiff > 180)
                        nHueDiff = abs(nHueDiff - 360);

                // Saturation and Brightness differences are checked together
                int nSDiff = abs(hsbFG.nSaturation - hsbBG.nSaturation);
                int nBDiff = abs(hsbFG.nBrightness - hsbBG.nBrightness);
                int nSBDiff = nBDiff + nSDiff;

                // Handle B/W colors differently
                // (since the HUE makes no difference)
                if(max(hsbFG.nSaturation, hsbBG.nSaturation) <= 5
                        && nSDiff < 6)
                {
                        if(nBDiff < 4)
                                bVisible = false;
                }
                else
                {
                        // If the FG hue is within 5 of the BG hue...
                        if(nHueDiff <= 5
                                && nSBDiff <= 13)
                        {
                                bVisible = false;
                        }
                }

                return bVisible;
        }
} // namespace ColorLogic
#endif // _COLOR_LOGIC_INCLUDE
```